Getting the angular position from gyroscope data

Submitted by Pieter-Jan on Thu, 06/09/2012 - 02:41

In this article I will explain how I succeeded in finding the angular position (angle) of one axis of a quadrocopter by integrating gyroscope data. For this article I will use the gyrosocope from the Hobbyking HK401B module I hacked in a previous article. Because I wanted a fast prototyping method I used an arduino, but the principle is the same on any other device. I will include the code for a PIC microcontroller as well.

A little math

Before we take a look at the program we will go through the basic mathematics involved here, as they will make understanding the program a lot easier.

The gyroscope gives us **the rate of change of the angular position over time** (angular velocity) with a unit of [deg./s]. This means that we get the derivative of the angular position over time.

$$\dot{\theta} = \frac{d\theta}{dt}$$

However rate feedback is extremely useful in control engineering, it is usually used in combination with position feedback. To obtain the angular position, we can simply integrate the angular velocity. So, assuming that at t=0 theta=0, we can find the angular position at any given moment t with the following equation:

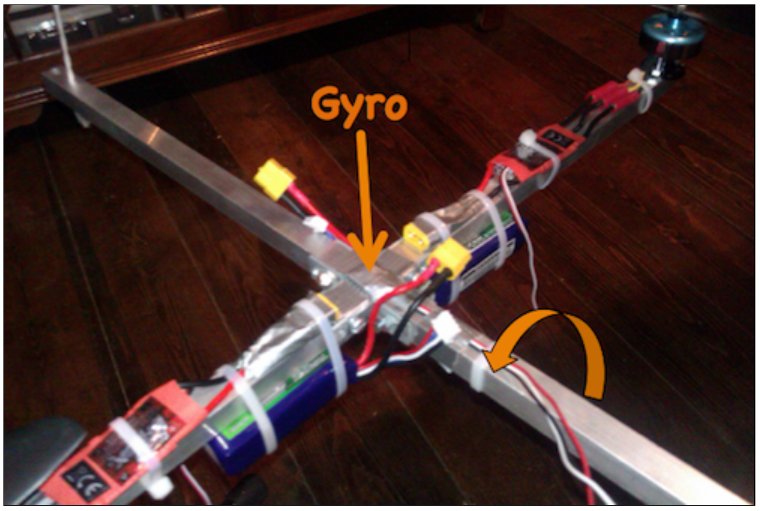
$$\theta(t) = \int_0^t \dot{\theta}(t)dt \approx \sum_0^t \dot{\theta}(t)T_s$$

The third part in this equation shows **the approximation we make when using digital systems**. Because we can't take a perfectly continuous integral, we have to take the sum of a finite number of samples taken at a constant interval Ts. Ts is called the sampling period. Of course this approximation will introduce errors. When gyrosocope data changes faster than the sampling frequency, we will not detect it, and the integral approximation will be incorrect. This error is called **drift**, as it increases in time. It results in the sensor reading not returning to 0 at the rest position. For this, it is important that we choose a good sampling period. The sampling frequency recommended for mechanical systems lies between 100 and 200Hz. I will use 100Hz (sampling period of 10ms) as this equals the cycle time of my quadrocopter PWM program.

Choosing a sampling frequency of 100Hz is a good choice for an "ideal case" mechanical system. It has to be understood though that **disturbance inputs** like shocks & vibrations because of the motors can have a much higher frequency and will not be measured properly. This is mostly the case for disturbances that work directly on the sensor, and it is a big problem of this integration method. Disturbances like wind and external forces that work on the system (quadrocopter) will be slowed down by the intertia of this system so that they will be measured correctly and can be compensated for.

The gyroscope sensor

This sensor was already discussed briefly in a previous article. It is an **analog sensor**, which means that it will output an analog voltage that will be a measure for the angular velocity. It is also a **one axis** gyroscope, which means that there is only one output. The axis that this sensor can measure falls together with the axis of the wires. It is shown in the picture below.



When the angular velocity is 0 deg./s, the sensor will output a voltage that is close to half of the supply voltage. A negative velocity will decrease this voltage while a positive velocity will increase this voltage. In the previous article I put a reference to a datasheet, but it doesn't seem to be correct. After 50 mails to Hobbyking, I gave up finding the datasheet, and just found out my values by trial/error. This what I came up with at a supply voltage of 5V:

- Offset (output at angular velocity = 0): **2.65V**
- Sensitivity (scale factor): **107.42 mV/deg./s**

The value for the offset can be measured with a multimeter. To find the value for the sensitivity, I wrote the arduino program that is shown below, and used the Serial monitor to check the correctness. E.g. hold the quadrocopter at 90 deg. and see if the measurement is correct or not. Trial/error! The weird number "107.42" was a result of the conversion from digital to analog.

The arduino program

Let's take a closer look at the arduino program now. If you are using the gyroscope from the HK401B like me, you can connect it by putting 5V between the black and red wires of the gyro (or whatever colors you soldered to the supply pins) and by connecting the white (signal) cable to the A0 analog input. If you are using another analog sensor, it should be easy enough to figure out how to connect it! The whole program works with 8 bit values. This means that the analog offset value of 2.65 results in a digital value of 136:

$$\frac{2.65}{5}256 \approx 136$$

The same principle would apply to the sensitivity if it was given in the datasheet. I found this value by trial/error so I knew it was 5.5 digitally.

However the arduino has a 10-bit ADC, I only use the 8 most significant bits. The last two bits of the ADC are usually not used because they contain a lot of noise and it is a lot more efficiënt to work with 8-bit values on an 8-bit processor ;).

It should be very easy now to see how I converted the approximated integral equation we saw earlier into an extremely simple program. The program runs at 100Hz. Every iteration we read the sensor, and convert its value into an 8-bit digital number (gyroValue). After this, the offset is removed and the gyroValue is being converted to [deg. /s] by multiplying it with the sensitvity. Finally we multiply it with the sampling period (*0.01 or /100) and add it to the angle, which is the sum (approximated integral) of all previous values.

It is very important to take a look at the datatypes you are using. I recommend using a float for the angle and the rate. The reason for this is that if you make more approximations by rounding off to another datatype, the drift would grow (extremely) fast. The only way this integral approximation will work, is by doing it very precise.

```
unsigned char gyroValue = 0;
unsigned char offset = 136;
float angle = 0;
float gyroRate;

void setup() {
  Serial.begin(9600);
}

void loop() {
  gyroValue = analogRead(A0) >> 2;   // Work with 8 most significant bits!

  gyroRate = (gyroValue - offset) * 5.5; // Sensitivity has been found trial/error
  angle += gyroRate / 100.0;

  Serial.print(angle);
```

```
  Serial.print("\n");

  delay(10);
}
```

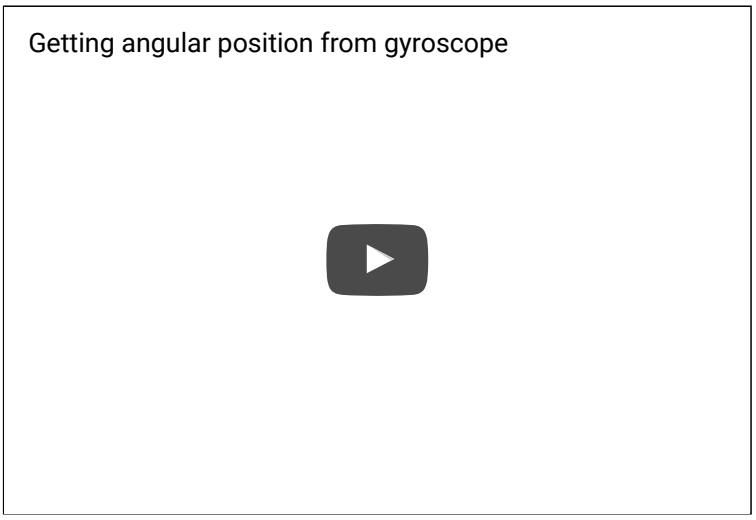
The PIC program

To demonstrate how easy it is to port this code on another platform, I included my implementation for the PIC16F690. A 20MHz crystal was used, and the gyro is connected the RA0-pin. I make use of the HI-TECH C Lite compiler.

```
#include <pic.h>
#include <pic16f690.h>

__CONFIG(FOSC_HS & WDTE_OFF & PWRTE_OFF & MCLRE_OFF & CP_OFF);

#define _XTAL_FREQ 20000000

/* Function prototypes */
void ADC(void);
void main(void);

unsigned char gyroValue;
unsigned char gyroOffset = 136;
float gyroRate;
float angle = 0;

/* Main program */
void main(void)
{
    // Init
    TRISA = 0b00000001;     // RA0 input (Analog)

    ADCON0 = 0b00000001;    // AD conversion init RA0 = input
    ADCON1 = 0b00000000;

    while(1)
    {
        // Read Gyro
        ADC();
        gyroValue = ADRESH;

        gyroRate = (gyroValue - gyroOffset) * 5.5;
        angle += gyroRate / 100.0;

        __delay_us(10000);      // 10 ms
    }
}

void ADC(void)
{
    GO_DONE = 1;
    while(GO_DONE);
}
```

Results

Here is a little video of the program in action. It is very clear to see the problem of **drift** introduced by approximating the integral by a sum. I also gave the sensor a few ticks with my finger to show that the drift is much stronger as a result of external forces that work directly on the sensor (e.g. motor vibrations, shocks, etc.).



UPDATE: To solve the drift error, we can combine the gyroscope data with accelerometer data using a **Complementary Filter**. This is described here.

Tags:
Quadrocopter